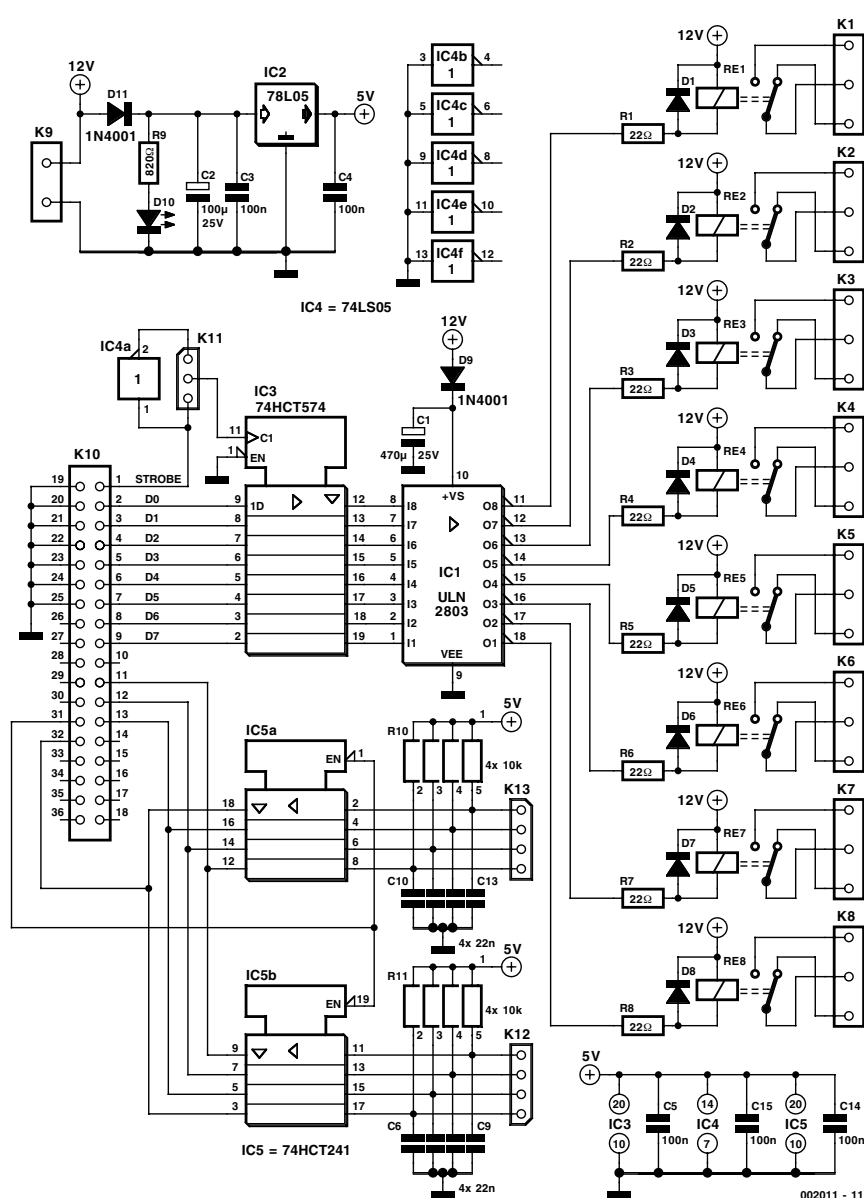


Universal Parallel Input/Output for PCs

Hardware design by D. Aggelos

Here's a simple project with Windows control software that allows you to control up to eight relays or other actuators, and read back an equal number of input lines, all by means of the parallel port on your PC.



The UPIO software and hardware (a board attached to the PC's parallel port) allows you control up to eight relay outputs using a Windows-style user interface designed for user-friendliness. UPIO also allows you read back logic states on input lines.

Circuit description

The circuit that belongs with the UPIO program is very simple and consists of a few low-cost and easy to find parts. The circuit diagram is given in Figure 1. A tri-state buffer type 74HCT241 (IC5) arranges all input contact reading. An 8-bit latch type 74HCT574 (IC3) is used to preserve the output state during instruction execution. The third essential component is a power driver type ULN2803 (IC1) which enables output relays Re1-Re8 to be actuated and de-actuated.

Because the LPT input port consists of just four bits (one nibble), two subsequent readings are required from the HCT241 to obtain the whole (8-bit wide) input word. Supply power comes from an external mains adaptor with an output of approximately 12 volts DC. An on-board regulator type 78L05 (IC2) provides the HCT and LS ICs with a 5-volt supply rail. Note that the ULN2803 and relays are powered from the unregulated 12-V supply. Obviously, the mains adaptor should be able to supply the necessary current (all relays may be actuated at the same time!).

Figure 1. UPIO circuit diagram.

The PCB

The PCB for this project was redesigned to reflect Elektor style and standards from artwork originally supplied by the author. The resulting design is shown in **Figure 2**. The ready-made board is available through our Readers Services as item 002011-1.

A PCB-mount 'blue ribbon' 36-way Centronics socket is used at the input to allow easy connection to your PC's printer port. At the 'output' side of the board, 3-way screw terminal blocks (connected directly to the relay contacts) allow the user to select N.C. (normally closed) or N.O. (normally open) type contacts. The digital inputs are divided over two 4-way SIL connectors, K12 and K13.

COMPONENTS LIST

Resistors:

R1-R8 = 22Ω

R9 = 820Ω

R10, R11 = SIL array 4 x 10kΩ

Capacitors:

C1 = 470μF 25V radial

C2 = 100μF 25V radial

C3, C4, C5, C14, C15 = 100nF

C6-C13 = 22nF

Semiconductors:

D1-D8 = 1N4148

D9, D11 = 1N4001

D10 = LED

IC1 = ULN2803

IC2 = 78L05

IC3 = 74HCT574

IC4 = 74LS05

IC5 = 74HCT241

Miscellaneous:

K1-K8 = 3 way PCB terminal block, raster 5mm

RE1-RE8 = PCB mount relay, 12V, e.g., Siemens V23040-A0002-B201

K9 = 2-way PCB terminal block, raster 5mm

K10 = 36-way Centronics connector, PCB mount

K11 = 3-way SIL pinheader with jumper

K12, K13 = 4-way SIL-header PCB, order code 002011-1, see Readers Services page and Elektor website.

Disk, contains all project software, order code 002011-11, see Readers Services page and Elektor website (free download for subscribers)

Tailor-made process control

Originally, the author supplied rather simple control software. By means of a 'program-style file', created by the application, it was possible to produce a simple

'Output Flow' with user-defined timing. In addition, output states could be checked by reading up to eight contact-type inputs. Together with the output control this allows 'Hold' or 'One-Shot' action to be implemented, both are well-established features in flow control systems.

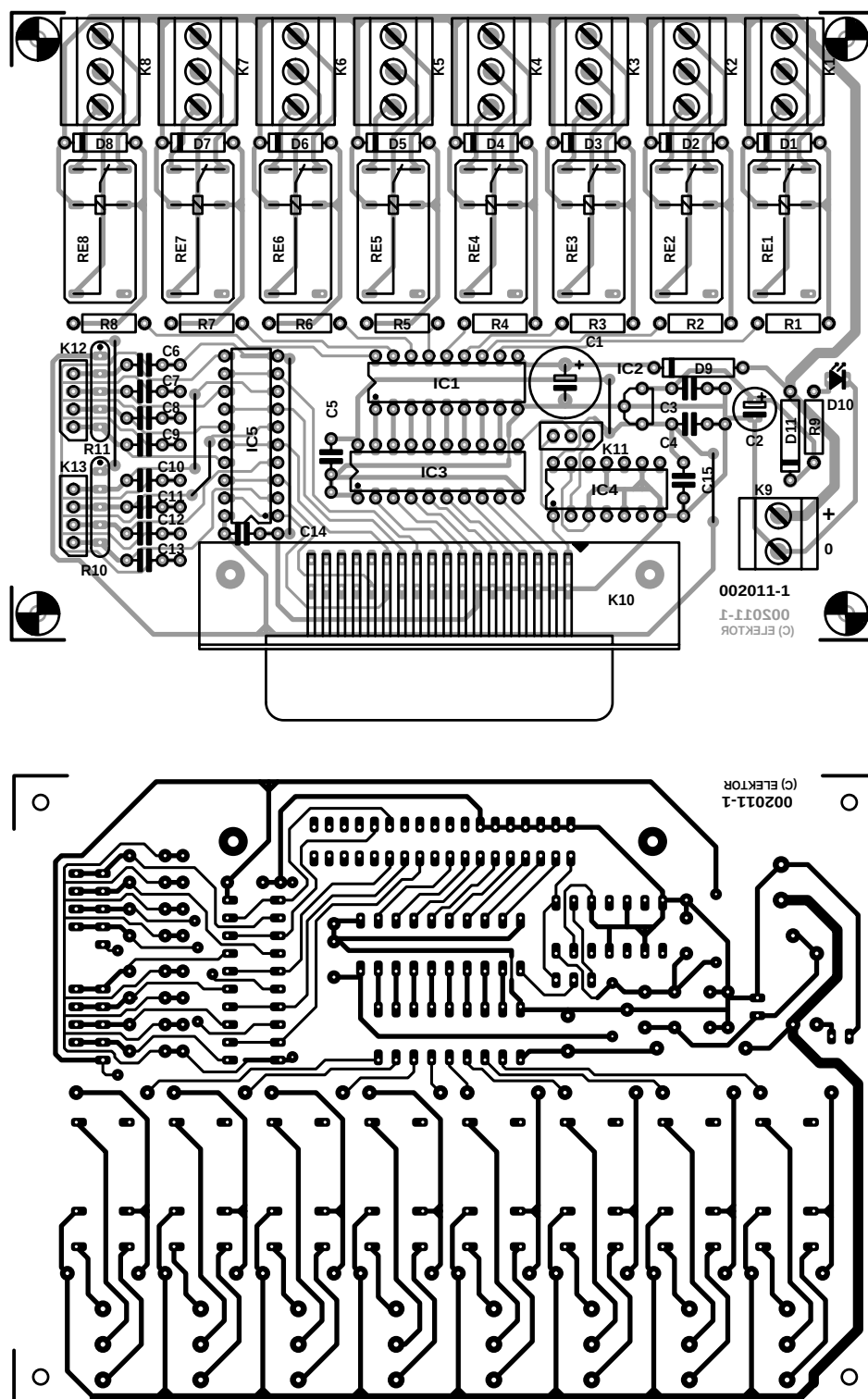


Figure 2. PCB design for the UPIO hardware (board available ready-made).

Talking to ports under Windows 95/98/NT

Having played around with the original software for a while it was felt that this was rather unrefined and unreliable. So our staff designer Luc Lemmens set out to write an improved control program for the UPIO hardware. Let's look at the underlying principles.

Under DOS and Windows 3.1, it used to be relatively easy to control PC hardware directly using higher programming languages like BASIC or Pascal. All that was needed at that time was the odd IN/OUT instruction in BASIC, or its PORT equivalent in Pascal.

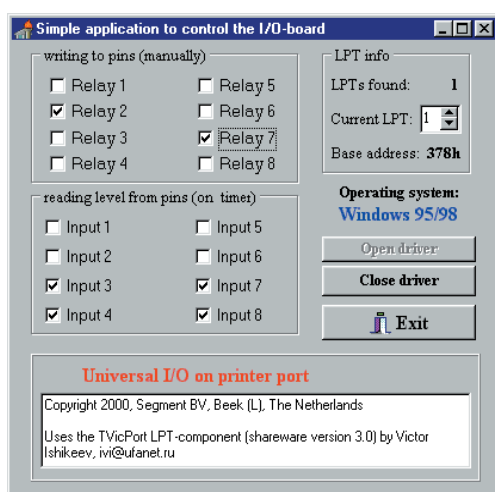


Figure 3. Screenshot showing the UPIO control software in action.

In later versions of the Windows GUI, (release 95 and later), the user is increasingly 'isolated' from the hardware, and it is no longer possible (and even forbidden under Windows 95) to communicate directly with various registers and memory locations. This kind of 'Smalltalk' is completely left to the operating system in conjunction with various device drivers. Writing your own device drivers is arduous to say the least, so it is often better to go on the Internet and see if someone else discovered the wheel for you. In this case we landed at www.entechtaiwan.com where we found a (shareware) LPT component called TVicPort written by Victor Ishikeev. This component is available in versions for Delphi (version 2 through 5), Java Builder (version 1, 3 and 4), Visual BASIC (version 6), Borland C++ and MS Visual C++ in Windows 95/98 and Windows NT.

As could be expected in this day and age of object-oriented programming, the printer port is no more than an *object* with all sorts of *properties* and responding to all sorts of *events*. Using this concept the programmer has easy access to the object (in this case the printer port), while control is exerted via *methods* associated with the relevant component. A brief discussion of the methods and prop-

erties used in the example program (a Delphi 2 application) will be given below.

The really essential procedures are *OpenDriver* and *CloseDriver* which, you guessed it, open and close the TVicport respectively. The logic variable (property) *ActiveHW* flags if the port is open or closed, or, in actual fact, if the driver is available or not.

The property *LPTNumPorts* indicates the number of printer ports in the system, while *LPTNumber* determines which LPT is being controlled by the driver. Finally, *LPTBaseAddress* shows the base address of that particular port. The component also has a property called *Pin*, an array of 25 logic variables (bits) representing the logic levels (states) found on hardware port pins. By means of this array programmers have easy access to the pins on the printer port. You only have to be cautious which pin is input or output only, or bi-directional.

HWTest and the example program

The HWTest program as supplied on the project disk illustrates the practical use of this component in many ways. In fact, we'd say it is a perfect starting point for your own applications.

Naturally the example program (also supplied on disk) was derived from HWTest. The 8 relays on the UPIO board may be switched on and off by ticking the relevant boxes. The ticks in the panel below the relay boxes indicate if a digital input is high or low. If the PC contains several printer ports, the thumbwheel switch allows a different port to be selected. As a matter of course, the printer port control will only work if the driver has been successfully loaded by pressing the 'Open Driver' button. A screenshot of the UPIO control program is shown in Figure 3.

Tools used for the project

For the original UPIO.exe application, the author used Borland Delphi development environment (Version 1-16 bits), which generates the '.dpr' application program files, the '.pas' unit files, the '.opt' and '.res' files. All original files for the UPIO project, as well as the refurbished version of the control software may be found on a floppy disk with order code 002011-11. It should be noted that the source code file as we received it from the author is supplied 'for information only' — it can

not be edited because the LPT component is missing. Also, because of various problems we do not recommend using the original '.exe' file.

As a matter of interest only, the circuit diagram was originally designed using 'Protel' schematic capture, while 'Advanced Schematics' software was used for the PCB design. Both the circuit diagram and the PCB artwork were redesigned by Elektor Electronics for the purpose of this article.

(002011-1)

Design and software editing: L. Lemmens
Text editing: J. Buiting